VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2017

Lukáš Unzeitig

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Lukáš Unzeitig**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Rieter CZ s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Marek Běhálek, Ph.D.**

Konzultant bakalářské práce: Ing. Pavel Kousalík

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 25. dubna 2017 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

Rieter CZ s.r.o.
Ústí nad Orlicí
(117)

V Ostravě 25. dubna 2017 .....

Abstrakt

Tato práce je výsledkem mé bakalářské praxe ve firmě Rieter CZ s.r.o., kde pracuji na pozici softwarového vývojáře. Náplní této praxe je pracovat na přidělených projektech.

Na začátku praxe jsem byl prověřen složitější úlohou, která měla ověřit moje nabyté znalosti a především schopnost učit se. Po určitém čase jsem dostal samostatný projekt, který zahrnoval tvorbu a následné testování jednotlivých komponent. Náročnost tvorby těchto komponent postupně rostla od jednoduchých vstupů až po složité a rozsáhlé navigace, grafy apod. Spolu s komponentami jsem tvořil také prezentaci, kde jsem rozebíral konkurenční software a analyzoval jeho nedostatky a výhody a následně navrhl vlastní řešení s připravenými ukázkami.

Klíčová slova: React, Flux, Rieter, komponenta, canvas, frontend

Abstract

This bachelor thesis is the result of my vocational practice in the firm Rieter CZ s.r.o. (Ltd.) where I am working as a software developer right now. The aim of this practice is to work on assigned projects.

I was authorised to solve a more sophisticated task at the beginning of my practice for the purpose of checking my gained knowledge and, above all, the ability to learn. After some time I received a separate project, which included the creation and subsequent testing of individual components. The complexity of creating these components increased gradually from simple to complex and extensive navigation, graphs, etc. Besides the above-mentioned I also made a presentation in order to evaluate our competitors' software. I tried to analyze its drawbacks and advantages and suggested my own solutions accompanied with prepared samples.

Key Words: React, Flux, Rieter, component, canvas, frontend

Obsah

Seznam použitých zkratek a symbolů

JS	–	JavaScript
WS	–	WebStorm
HTML	–	Hyper Text Markup Language
CSS	–	Cascading Style Sheets
SW	–	software
DOM	–	Document Object Model
MVC	–	Model-view-controller
UI	–	user interface

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Pro mou bakalářskou praxi jsem si vybral firmu Rieter CZ s.r.o., která patří k jedněm z největších firem v České republice. Firma má sídla po celém světě od Ameriky počínaje přes Indii až po Čínu. Tato firma nabízela pracovní pozici na místo webového vývojáře. Tato pozice požadovala znalosti hlavně JS, HTML a CSS. Dále znalost angličtiny na pokročilé úrovni, schopnost pracovat v týmu, učit se a tendenci zlepšovat se.

V první části mé práce bych rád popsal firmu Rieter a mé pracovní zaměření. Ve druhé popíšu JS knihovnu React, její základní strukturu a architekturu Flux. Ve třetí pak projekty, na kterých jsem dělal a dělám, a uvedu seznam všech zadaných úloh, které jsem dostal k řešení. Dále tyto úlohy podrobně popíšu, jak problém takový, tak důvod zvoleného řešení a následnou samotnou implementaci. K některým úlohám doložím ukázky kódů, případně obrázky výsledných funkcí/vizualizací. Ke konci zhodnotím svůj pokrok, nabyté vědomosti a využité znalosti ze studia na škole, a to jak teoretické tak praktické. Také se zmíním o nedostatcích, které mi v průběhu praxe chyběly. Na úplném konci celkově zhodnotím mou bakalářskou praxi.

2 Popis firmy Rieter CZ s.r.o.

2.1 O společnosti

„Společnost Rieter CZ s.r.o. se sídlem v Ústí nad Orlicí je součástí švýcarského koncernu Rieter od roku 2001, který je předním světovým výrobcem strojních zařízení a ucelených systémových řešení pro textilní průmysl.

Rieter CZ s.r.o. charakterizuje silný důraz na inovace, které jsou realizovány ve vlastním vývojovém centru. Dalšími charakteristickými rysy jsou zaměření na přesnou strojírenskou výrobu textilních strojů a strojírenských komponentů při použití moderních technologií a procesů, výrobu řídících elektro rozvaděčů a kabelových svazků.

Společnost klade důraz na vysoký stupeň zákaznické orientace a budování dlouhodobých vztahů s našimi zákazníky. Plná integrace do struktur koncernu Rieter, využití zahraničního know-how a zaměření na stabilní růst dávají Rieter CZ s.r.o. velmi dobrou perspektivu dalšího rozvoje."[2]

2.2 Popis pracovní pozice

Pracuji v týmu Vývoj elektroniky, ve kterém je 24 lidí, a ten aktivně pracuje na zákaznických projektech. Od začátku praxe jsem pracoval na projektu HMI tester, který měl zahrnovat tvorbu nového uživatelského webového rozhraní pro ovládání textilních strojů. Má první úloha měla vyzkoušet mou schopnost učit se. Zahrnovala naučení se JS knihovny React, ve které jsem implementoval komplexní matici (graf).

Druhá úloha zahrnovala naučení se a následnou implementaci architektury Flux, která měla reprezentovat lokální uložiště, ve kterém se udržovala veškerá data ze stroje.

Další úlohy zahrnovaly tvorbu menších jednotlivých komponent/funkcionalit, jako jsou vyskakovací modální okna, numerická klávesnice, jazyková mutace a podobné , které jsem postupně implementoval do projektu.

Poslední a nejkomplexnější úlohou bylo analyzování konkurenčních SW a následné navrhnutí a implementace vlastní navigace, které mělo být intuitivní, přehledné a minimalizované oproti původnímu.

3 Využité technologie

3.1 React

Jedná se o JS knihovnu, která pomáhá při tvorbě interaktivního uživatelského prostředí a slouží pro vytváření webových komponent. V pomyslném MVC reprezentuje právě "V"neboli view vrstvu.

React používal a vyvíjel Facebook, který ho roku 2013 vypustil ven jako opensource. K dnešnímu dni patří tak k jedněm z nejoblíbenějších a nejaktivnějších repozitářů na GitHubu.

React efektivně aktualizuje a renderuje komponenty, které změnily svůj stav a ostatní nechává být, čímž velmi zrychluje chod celé aplikace. Přesněji, při změně stavu nějaké komponenty vytvoří nový virtuální DOM, který pomocí chytrých algoritmů porovnává se stávajícím a při nějakém rozdílu překreslí jenom změněnou část.

Ukázka kódu číslo [1] představuje základní komponentu, která má vlastní konstruktor a ten může a nemusí být implementovaný. Pokud konstruktor nevytvoříme, vytvoří se sám zcela prázdný tak, jako tomu je i v jiných programovacích jazycích, jako je Java,C# apod. Dále tu máme metodu render, která je povinná. Už jen z názvu je jasné, že tato metoda se stará o vykreslování všech elementů a dat. Další důležitá metoda se jmenuje *setState*. Právě uvnitř této metody můžeme nastavovat data dané komponenty. K tomu, aby se komponenta vykreslila do DOMu, je důležité, aby se na zaváděcí metodu zavolalo *ReactDOM.render* a jako první parametr se předala právě tato zavádějící komponenta a jako druhý parametr element, do kterého chceme tuto komponentu vykreslit.

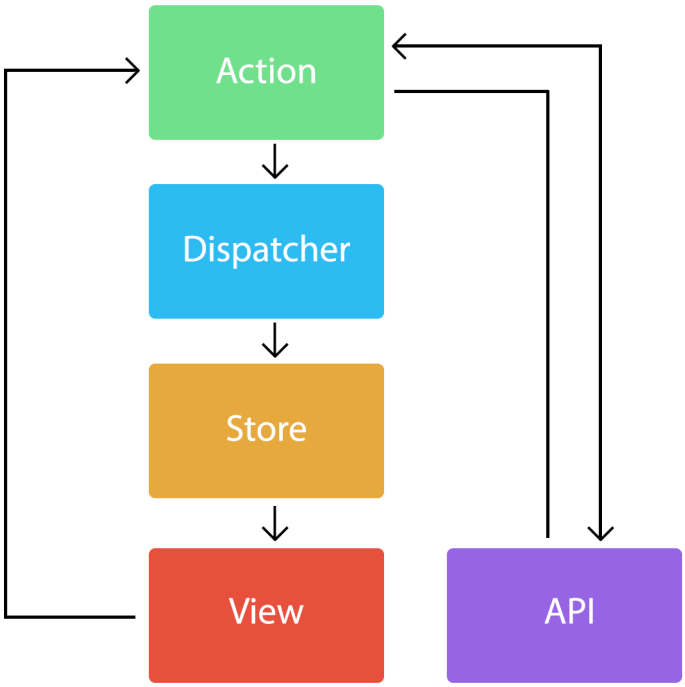
```
class App extends React.Component{
    constructor(props){
        super(props);
        this.state = {
            a: 0
        }
    };
    render(){
        return(
            <div> Hodnota a je : {this.state.a}</div>
        )
    }
}
ReactDOM.render(<App />, document.getElementById("app"));
```

Výpis 1: Ukázka základní React komponenty psané v JSX

3.2 Flux

Flux je aplikační architektura, kterou používá Facebook pro vytváření client-side webových aplikací. Skládá se ze tří části - action, dispatcher, store. Aplikuje se při návrhu jednosměrného i obousměrného toku dat v UI. Vychází z předpokladu, že výsledné UI je reprezentovatelné pomocí dat, která do něho přichází z uložišť.

Interakce s uživatelem se děje pomocí akcí. Tyto akce zpracovává tzv. Dispatcher, který následně informuje o změnách všechna uložiště a je na implementaci daného uložiště, jestli následující akci eviduje nebo ne. Tato uložiště potom poslouchají jednotlivé komponenty, které z nich berou data. Na změnu dat v uložišti reagují zavoláním metody *getData*, čímž způsobí překreslení této komponenty s novými daty. Na následujícím obrázku [1] je znázorněno schéma principu této architektury.



Obrázek 1: Schéma architektury Flux

4 Řešené projekty a úlohy

Tato část zahrnuje popsání projektů a seznam všech zadaných úloh.

4.1 Popis mého hlavního projektu

Mým hlavním úkolem bylo navrhnout a vytvořit nové webové uživatelské rozhraní pro ovládání textilních strojů a to takové, aby bylo hlavně intuitivní, přehledné, jednoduché, moderní a rychlé. Toto rozhraní mělo být implementováno pomocí React frameworku a mělo být spojené s Flux architekturou.

Původním rozhraním byla desktopová aplikace, která běžela na operačním systému Quenix. Kvůli rychlosti, omezené platformě a přístupu je toto rozhraní nahrazeno právě webovým, díky čemuž bude funkční na každé platformě, zrychlí chod a zároveň snadněji umožní vzdálený přístup přes síť, ať už třeba přes mobil nebo jiný počítač.

V průběhu tohoto projektu jsem dostal ještě jeden úkol pro chystaný firemní Den kariéry. Měl jsem vytvořit loterii, což bylo animované kolo, které se roztáčelo vzdáleně přes mobil a losovalo ceny. Na tomto projektu jsme pracovali v týmu dva. Já pracoval na webové aplikaci, která se starala o samotnou loterii a také sledovala stavy serveru, na které reagovala a po nějaké akci posílala nová data zpět. Kolega pracoval na mobilní aplikaci a samotném serveru.

4.2 Seznam zadaných úloh

Název úlohy	**Čas[h]**
Vytvoření matice pro zobrazení dat ze stroje(graf)	112
Vytvoření a připojení Flux architektury k projektu	16
Vytvoření funkcionality pro jazykovou mutaci	6
Vytvoření komponent vstupů, číselné klávesnice a následné propojení	24
Vytvoření univerzální komponenty modálního okna	8
Analýza konkurenčního SW	24
Návrhy možných navigací	60
Implementace navržených navigací	80
Tvorba prezentace návrhů	4
Tvorba sekundární navigace	20
Tvorba komponenty pro rychlé přepínání nastavení	20
Navrhnutí a tvorba ukázkové zobrazovací stránky	8
Tvorba komponenty lišty	25
Tvorba dynamicky se měnícího startovacího tlačítka stroje	6
Tvorba aplikace loterie	60

Tabulka 1: Přidělené úlohy a jejich časová náročnost

5 Zadané úlohy a postup jejich řešení

5.1 Překlady jazykových verzí

Zadáním této úlohy bylo vytvořit jazykovou mutaci pro 6 jazyků. S mutací v Reactu jsem zkušenosti neměl, tak jsem hledal řešení na internetu a na jedno narazil. Řešením bylo vytvořit json soubor, který můžete nalézt na ukázce kódu [2] , kde lze vidět definici jazyků a překladů slov, které se nachází na webu. Nalezené řešení fungovalo s Reduxem, což je architektura odvozená od Flux, tudíž jsem musel kód poupravit, aby fungovalo právě s mým Fluxem .

```
[
  {
  "lang": "cz",
  "page": {
       "menu": { "settings": Nastaveni", "product_setting": "Nastaveni produktu"},
         }
  },
  {
  "lang": "en",
  "page": {
       "menu": { "settings": Settings", "product_setting": "Product settings"},
         }
  },
]
```

Výpis 2: Ukázka json souboru

Při přepnutí jazyka jsem vyvolal akci, která mi načetla json soubor a vyfiltrovala podle chtěného jazyka přímo objekt, který obsahoval data celé stránky v požadovaném jazyce. Toto je vidět na následující ukázce kódu [3]. Data se poté uložila do uložiště a poslala do celé aplikace, kde si každá komponenta vzala potřebná slova z objektu. Výsledné řešení bylo nakonec velmi lehké, jasné a hlavně rychlé.

```
const content = require('../data/content.json');
var page_data = content.filter(obj => obj.lang === language)[0];
```

Výpis 3: Import jsonu a následná filtrace

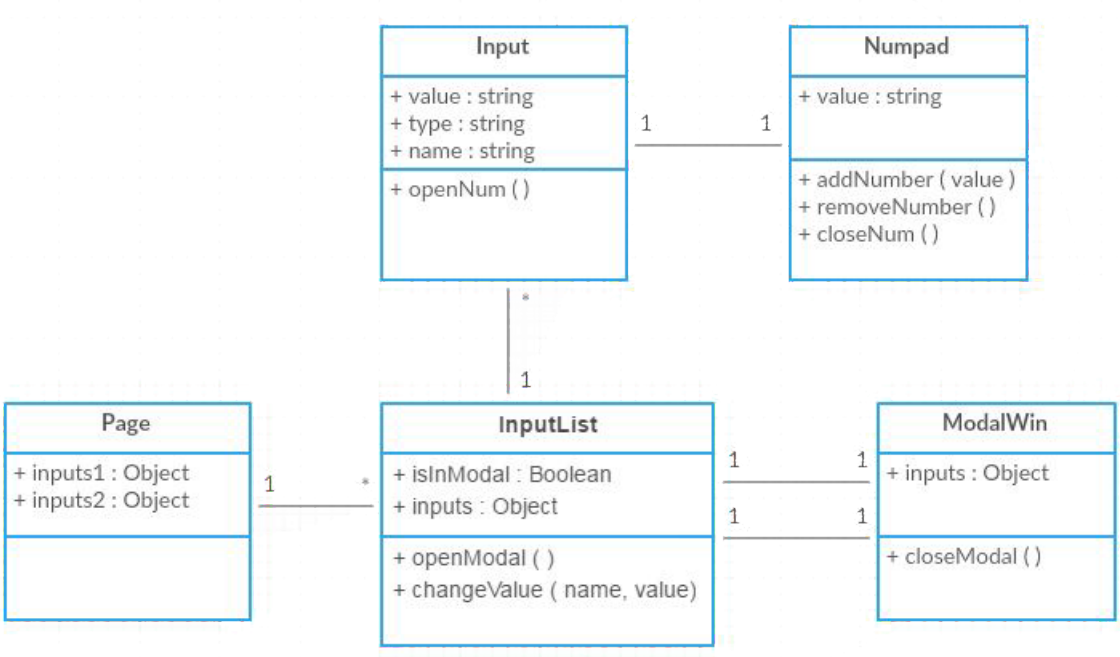
5.2 Vytvoření komponent vstupů, číselné klávesnice a následné propojení

Na první pohled se zdá, že tuto úlohu nemá smysl vůbec popisovat s ohledem na její trivialitu. Avšak není tomu tak. Vytvořit naprosto univerzální komponentu, která se vytvaruje, vykreslí a naplní pokaždé správně, ať je do ní zaslán objekt s dvěma vstupy a jedním jménem nebo dvěma jmény nebo se zaškrtávacím políčkem, není vůbec snadné. Navíc propojení s modálním oknem a číselnou klávesnicí dodává této úloze ještě více na obtížnosti.

Princip funkčnosti této úlohy má být takový, že po kliknutí na určité pole vstupů se otevře modální okno, ve kterém se zobrazí to stejné pole, jen větší, aby byly jednotlivé vstupy lépe editovatelné. V tomto zobrazení se při kliku na určitý vstup otevře číselná klávesnice, kde můžeme zadat hodnotu do daného vstupu. Po editaci všech chtěných hodnot můžeme modální okno potvrdit, změněné hodnoty se upraví v původním objektu a tento objekt se následně pošle na server.

Vzhledem k tomu, že list objektů, který se měl zobrazit v tabulce vstupů, byl pokaždé jiný, ať už se jednalo o chtěné elementy jako zaškrtávací pole, čistý vstup nebo název vstupu či jeho jednotku, mým cílem bylo vytvořit co nejmenší počet univerzálních komponent.

Na obrázku číslo [2] je znázorněna funkční, mnou navržená implementace řešeného problému.



Obrázek 2: Třídní diagram pro zadávání a ukládání hodnot do vstupů

5.3 Tvorba matice pro zobrazování dat ze stroje

Zadáním této úlohy bylo vytvořit komplexní matici, která se chová zároveň jako graf, který se vykresluje podle hodnot zadaných ze vstupů. Samotné buňky celé matice potom uchovávají dvě hodnoty, které chodí ze serveru a nedají se v aplikaci měnit.

Implementace byla jasná, každá buňka musela být samostatný objekt, který udržoval vlastní stav jako pozici, příchozí hodnoty ze serveru, barvu, velikost apod. Dále bylo potřeba implementovat sloupce, které reprezentovaly graf a musely být samostatné objekty, aby udržovaly svoji pozici, šířku a výšku.

Otázkou bylo, jakou technologii na vykreslování zvolit. Byla možnost použít samostatné divy, u kterých by byl ale velký problém s pozicováním. Reálným řešením tedy mohlo být buď použití SVG nebo canvasu. Po delším zvažování a hledání jsem zvolil JS HTML5 canvas knihovnu fabricjs, která umožňuje efektivně pracovat s tvary jako objekty. Dále umožňuje solidní interakci, volné kreslení, stylování i animace.

Samotný kód je velmi rozsáhlý. Obsahuje komponenty pro osy x, y, pro jednotlivé sloupce grafu, pro všechny sloupce, pro jednotlivé buňky matice, pro celou mříž těchto buněk a pak i pro celou matici. Na obrázku číslo [3] je znázorněn třídní diagram těchto komponent. Je to celkem přes 500 řádků kódu. Tvorba této celé komponenty mi zabrala asi nejvíce času, hlavně kvůli složitosti. Všechno se týkalo matematiky, dopočítávání velikostí. Největší práci jsem měl se zjištěním algoritmu pro dopočítávání pozice dalších sloupců a to z důvodu, že požadovaná hodnota např. 50 neodpovídala 50ti bodům v canvasu. Nesrovnalost způsobily hodnoty na osách, které se nepravidelně inkrementovaly (2, 10, 20, 30, 50, 80, 160, 320, 500),(-30, -20, 0, 20, 40, 16, 80, 100, 120, 200). Přitom mezera mezi těmito hodnotami v grafu byla vždy o velikosti 70 bodů. Algoritmus lze vidět na následující ukázce kódu [4].

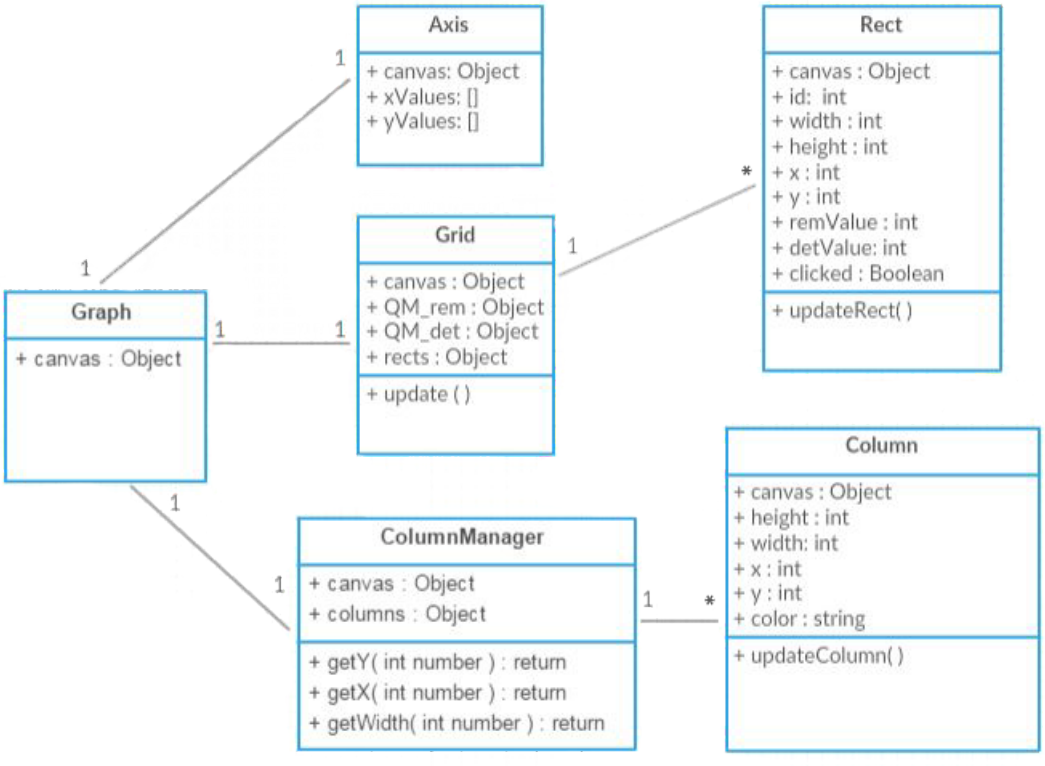
```
getX(a){
    var numbers=[2, 10, 20, 30, 50, 80, 160, 320, 5000];
    var i = 0;
   var cellWidth = 70;
    var number = 0;
    for(;i<numbers.length;i++){
      if (numbers[i] >= a)
        break;
    }
    if(i>=4)
        number = ((a-numbers[i-1])/(numbers[i]-numbers[i-1]) *(i*cellWidth -(i-1)*cellWidth)+(i
            -1)*cellWidth);
    else
      number =(a/numbers[i])*(i*cellWidth);
    return Math.round(number);
}
```

Výpis 4: Algoritmus pro převádění zadané hodnoty x do reálných hodnot v canvasu

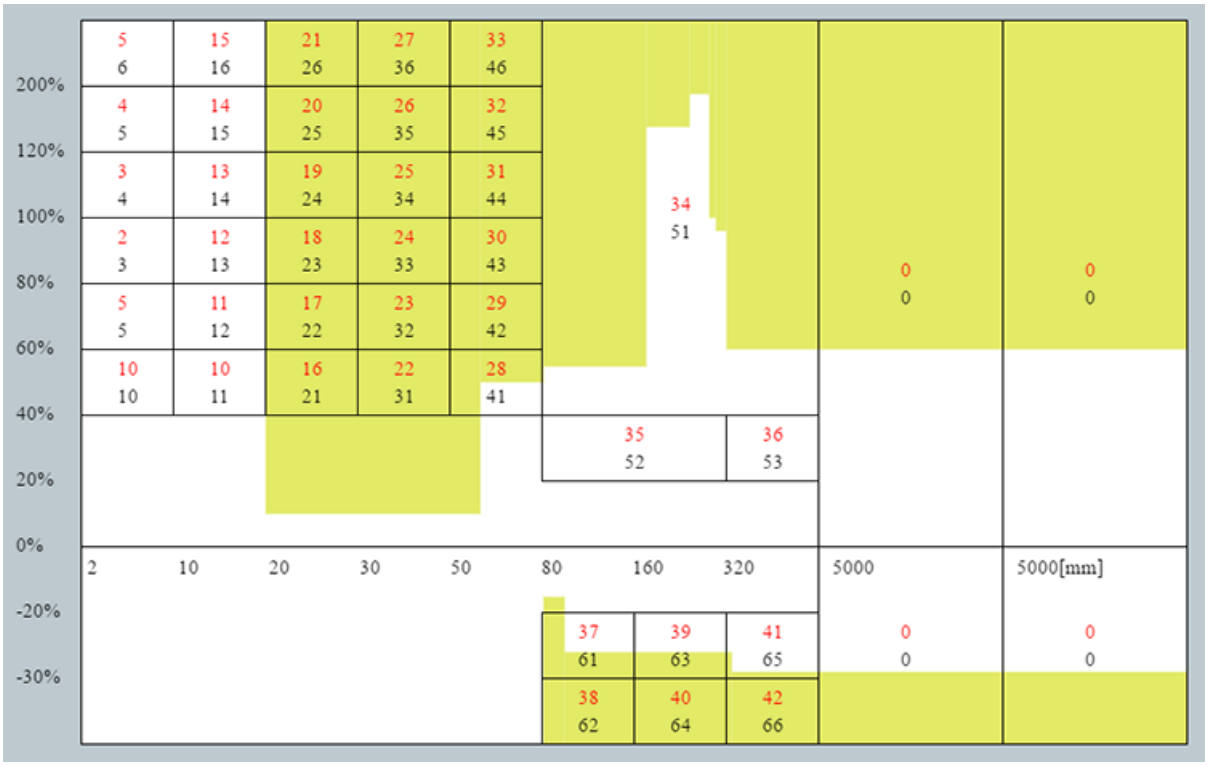
Při implementaci této komponenty jsem měl hned několik problémů. Ze začátku to byla samotná spolupráce fabricjs s Reactem, i když React je vlastně JS knihovna a měla by spolupracovat se všemi jinými JS knihovny. Vždy tomu tak úplně není. Snaha spustit client-side knihovnu na serveru se z počátku nedařila. Dále vznikal problém při posílání odkazu na renderovaný canvas, neboť jsem nemohl vzít odkaz doposud nevykresleného canvasu, to bylo možné až po prvním vykreslení. Musel jsem tedy vyvolat další render celé komponenty, ve kterém se konečně přeposlala reference vytvořeného canvasu všem potomkům. Implementace tohoto problému mi zabrala mnoho času, neboť jsem s Reactem začínal a to i přesto, že řešením tohoto problému bylo jen pár řádků kódu.

Dalším menším problémem bylo renderování sloupců na buňky matice, které měly bílou barvu. Požadavkem bylo, aby barva sloupců překrývala bílou barvu matice až na text a rámečky těchto buněk. Jedinou možností bylo nastavení průhlednosti sloupců, aby byly vidět zmiňované texty a rámečky. Nastavení průhlednosti tyto předměty jemně obarvovalo, což nebylo pěkné. Řešením toho problému bylo vytvoření bílého podkladu pro celou matici, na který se renderovaly barevné sloupce a na ně pak jednotlivé buňky matice, které měly výplň zcela průhlednou.

Po vyřešení těchto problémů šla implementace celkem snadno. Na obrázku [4] lze vidět výslednou vizualizaci.



Obrázek 3: Třídní diagram



Obrázek 4: Ukázka matice

5.4 Návrh a implementace rozsáhlého menu

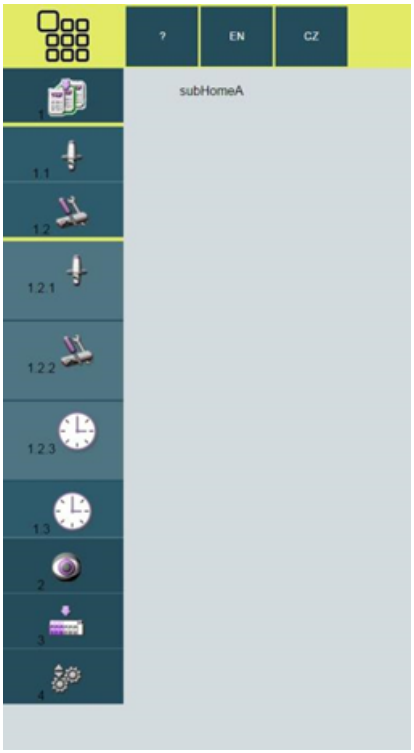
5.4.1 Návrh

Po větších zkušenostech s Reactem jsem dostal zadáno vytvořit a navrhnout menu. Aktuální používané menu na strojích je až čtyřúrovňové, kde pod jednou položkou může být až 8 dalších podpoložek. Navíc struktura celého menu se mění podle přihlášeného uživatele. Pokud stroj obsluhuje zrovna běžný zaměstnanec, nabídka je jiná, než když je u stroje zaměstnanec, který ho spravuje. Každý zaměstnanec se ke stroji přihlašuje klíčem, který ho identifikuje.

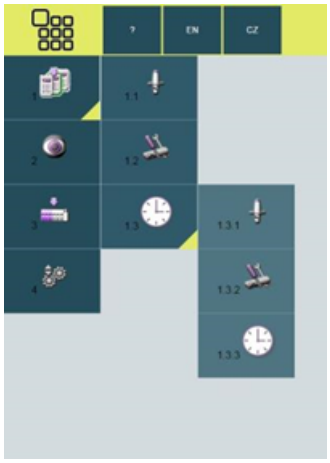
Cílem tedy bylo vytvořit dynamické menu, které mělo umožňovat co nejrychlejší přístup k jakékoliv položce. Mělo být intuitivní, jednoduché a přehledné. Dalším cílem bylo toto menu minimalizovat a to do maximální hloubky tří úrovní.

Začala tedy tvorba návrhu menu. Navrhnout takové menu, které by splňovalo všechny tyto požadavky, nebylo vůbec snadné. Hlavní problém se týkal velikosti obrazovky, kde panely, na kterých běží tato aplikace, mají poměr stran 4:3, rozlišeni 1024 x 768 a velikost displeje pouhých 15 palců. Jelikož se jedná o dotykové displeje, položky musí být tak velké, aby se na ně dalo jednoduše kliknout. Mé první dva návrhy se s tímto problémem dosti potýkaly.

Na obrázku [5] je vidět můj první návrh čistě horizontálního menu, které se s problémem potýkalo nejvíce. Na obrázku [6] je vidět druhé řešení, do kterého jsem implementoval i vertikální část menu. Toto řešení redukovalo velikost menu, ale pouze částečně.



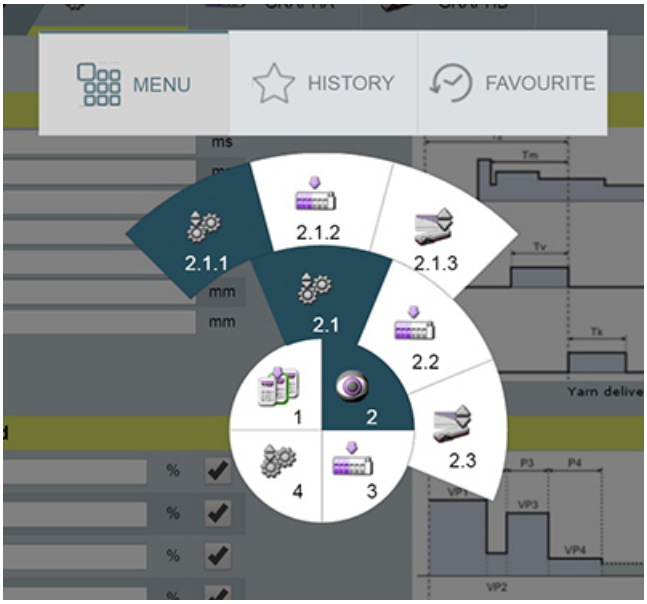
Obrázek 5: Ukázka první navigace horizontálního menu



Obrázek 6: Ukázka druhé navigace horizontálně/vertikálního menu

Pro názornou ukázku problému uvedu příklad. Při zvolení čtvrté položky první úrovně menu se posouváme na čtvrtou pozici ze shora. Podpoložek této položky může být až 7, tudíž se dostaneme až na 11. pozici ze shora. Tato pozice je téměř až dole na obrazovce a přitom třetí úroveň, která by mohla mít až 7 položek, je stále před námi. Řešením tohoto problému je vytvoření modálního vyskakovacího okna, které šetří místo, když je schované a naopak otevřené může být přes celou obrazovku. Pro toto řešení jsem navrhl dvě odlišná menu.

Obrázek 7: Ukázka první navigace čtvercového menu v modálním okně



Obrázek 8: Ukázka druhé navigace kruhového menu v modálním okně

Na obrázku číslo [7] je navržené čtvercové menu, který bylo mým prvním pokusem. Tento návrh je přehledný a velice intuitivní. Na úvodní obrazovce menu zobrazuje všechny položky druhé úrovně, protože první úroveň je pouze rozcestník, který otevírá další položky. Při rozkliknutí další úrovně se ve čtverci ukáží pouze položky navštívené úrovně. Pro možnost rychlé cesty zpět se do horní levé strany přidávají odkazy předchozích navštívených položek. Vpravo nahoře je možnost rychlé cesty zpět, vpřed, otevření naposledy navštívených nebo oblíbených položek. Toto menu má jedinou nevýhodu a to, že při hlubším procházení úrovní jsou vidět pouze položky aktuální úrovně, což zhoršuje rychlý přístup k jiným.

Řešení, které je vidět na obrázku [8], řeší veškeré požadavky na navigaci. Díky svému tvaru, může obsahovat velké množství položek, v první úrovni 4, ve druhé 8, 12 atd.. Řeší otázku rychlého přístupu, kde při návštěvě třetí úrovně jsou vidět všechny položky předchozích navštívených úrovní. Menu má navíc moderní kruhový design a intuitivní ovládání. Dále je zde možné zobrazení naposledy navštívených a oblíbených položek stejně jako u předchozího menu.

5.4.2 Implementace

Implementace takového kruhového menu byla velmi složitá. Hlavní otázkou bylo, jak řešit vykreslování položek menu. Prvním pokusem bylo namalování části kruhu v editoru. Tuto část jsem poté nastavil jako `background-image` každému divu, který jsem posouval kolem středu a div následně rotoval. Toto řešení bylo velmi nepřesné a namáhavé, zejména hrát si s každým pixelem při posouvání a tvořit takový tvar. Navíc nešla efektivně měnit barva aktivní položky, neboť se jednalo o obrázek.

Dalším řešením bylo stylování samotného divu, toto řešilo problém s obarvováním aktivních položek, nicméně vytvořit přesně chtěný tvar bylo ještě těžší, než ho ručně malovat. Dělat něco tak komplexního pouze v CSS není vůbec snadné, a proto bych to k řešení takového problému nedoporučoval. Jedinou možností bylo použít něco jiného než klasické stylování divů pomocí CSS.

Vybral jsem si proto značkovací jazyk SVG, který zvládá tvorbu takovýchto tvarů a to celkem jednoduše. Stačí pomocí svg elementu `<path>` vytvořit takový tvar, do kterého se přidá ještě text a ikona, pak určit jeho pozice a vykreslit. Tomuto elementu se dá za běhu programu měnit barva, velikost i pozice. Pozice jsem měl pevně definované, jen jsem si je přeposílal při rekurzivním volání další úrovně.

Kdybych tuto komponentu měl řešit znovu, vytvořil bych pouze první zavádějící pozici a pak bych už jen ostatní kopíroval a rotoval podle středu kruhu. Tento způsob jsem použil v jiné části projektu.

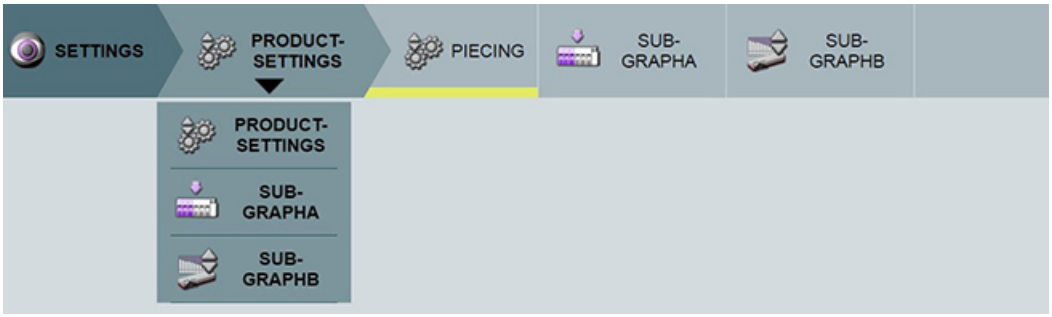
Řešení dynamicky se měnícího menu bylo díky inspiraci z jiného projektu celkem jednoduché. Vytvořil jsem si json soubor, kde jsem definoval celou strukturu menu, kde každý objekt měl svůj název stránky, ikonu, číslo a cestu. Při kliknutí na položku v menu se tento soubor prošel, vyhledala se požadovaná stránka a ta se poslala do zavádějící komponenty. Následně při tvarování kruhového menu se díky znalosti aktuální cesty navštívené stránky procházela struktura tohoto jsonu a mohla se tak formovat komponenta kruhového menu, která se vždy rekurzivně volala pro každou další navštívenou úroveň. Ukázku json souboru, který uchovával strukturu celého menu, lze vidět na ukázce kódu [5].

```
"settings": {
       hash: "/settings",
       name: "settings",
       page: null,
       value: "2",
       icon: "../../assets/Overview.png",
       children: {
           "product-settings": {
               hash: "/settings/product-settings",
               name: "product-settings",
               page: "ProductSettings",
               value: "2.1",
               icon: "../../assets/config.png",
               setting: ["shift","unit","group"],
               children: {
                   "piecing": {
                       hash: "/settings/product-settings/piecing",
                       name: "piecing",
                       page: "Piecing",
                       value: "2.1.1",
                       icon: "../../assets/config.png",
                       setting: ["shift","unit"],
                       children: {}
              },
           },
        },
     },
}
```

Výpis 5: Ukázka struktury celé navigace v json souboru

5.5 Návrh a řešení sekundární navigace

V předchozí úloze jsem tvořil navigaci, kterou bylo možno používat až po otevření modálního okna. Chtěl jsem tudíž vytvořit ještě něco jednoduchého a přehledného, co by tuto navigaci doplňovalo a co by bylo permanentně zobrazeno na obrazovce a přitom nezabíralo moc místa. Bylo také důležité, aby by byl možný přechod na jinou položku v aktuální úrovni a zároveň možnost chodu zpět. Pro toto řešení jsem zvolil horizontální menu, do kterého jsem časem implementoval i vertikální, které rozšířilo jeho využití a to tak, že umožňovalo volný pohyb v aplikaci a to celkem rychle.



Obrázek 9: Sekundární horizontální + vertikální menu

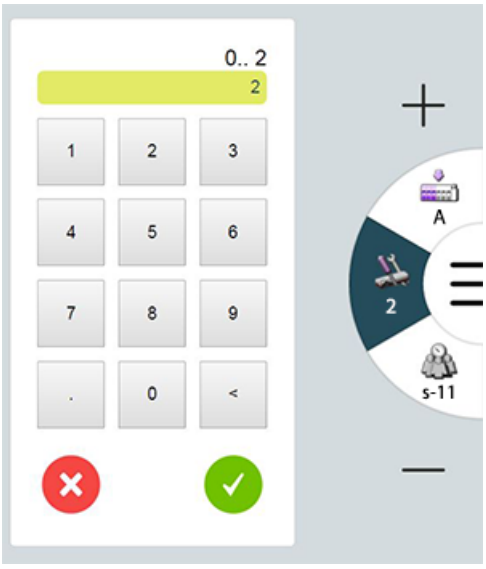
Na obrázku číslo [9] je zachyceno navržené řešení. V tomto návrhu je možnost rozbalení předchozí navštívené položky, kde se při kliknutí zobrazí celá předchozí úroveň, a je tak možné rychlé přesměrování jinam. Zároveň je díky odlišnosti barev úrovní ihned vidět, v jaké hloubce jste a které položky patří k sobě. Aktivovanou položku jsem pak ještě odlišil žlutým podtržením.

K vykreslování takového menu jsem kombinoval prvky klasického stylování divů a SVG prvků.

5.6 Řešení přepínání rychlého nastavení důležitých dat

Jednou z nejdůležitějších věcí v celé aplikaci, na které pracuji, je možnost rychlého přepínání mezi důležitými daty, jako je výběr směny, skupiny strojů, jednotek nebo přednastavených konfigurací. Proto řešení mělo být výrazné, přehledné a hlavně intuitivní, aby umožňovalo efektivně tato data měnit. Předchozí řešení v desktopové aplikaci bylo velmi nepřehledné a složité na pochopení. Změna měla přijít v nové aplikaci, kterou vyvíjím právě já.

Při řešení jsem se inspiroval u hlavního menu, které je kruhového tvaru. Navrhl jsem půlkruh, jehož části umožňují aktivovat nastavení, které chceme měnit, a středovým kolem se dále otevírá možnost této změny. V určitém případě se otevírá list, který umožňuje vybrat si určitou směnu nebo list nastavení apod., a nebo pak v případě možnosti volby jednotky se otevírá číselná klávesnice, která umožňuje zadávat číselné hodnoty. Na obrázku [10] je vidět výsledná vizualizace. Je zde otevřená číselná klávesnice pro zadání čísla jednotky.



Obrázek 10: Ukázka otevřeného číselného vstupu u vybrané volby pro změnu jednotky

Implementace potom vypadá podobně jako u kruhového menu, je jen jednodušší. Pro vykreslování všech položek v kruhu se vytvoří pouze jedna část kruhu, která se vždy jen okopíruje od předchozí, poté se pootočí o potřebný počet stupňů, který se spočítá z počtu možností nastavení / 180. Začíná se vždy od shora. Informace o tom, která obrazovka má jaké možnosti nastavení, je uložena ve vlastnosti objektu v json souboru, kde je uložena celá struktura navigace.

5.7 Tvorba komponenty lišty

Další zadanou úlohou bylo vytvořit něco, co umožňovalo rychlé přepínání mezi obrazovkami, ať už se jednalo o oblíbené nebo naposledy navštívené. Prvním řešení, o které jsem se pokusil, bylo zakomponovat do modálního okna s klasickým menu taková menu, která právě tyto položky zobrazovala. Pro přepínání zobrazených menu jsem zvolil tlačítka. Toto řešení zůstalo implementované v projektu, ačkoliv se nejednalo o příliš rychlé řešení, jelikož se pro přepnutí na jinou obrazovku musela většinou provést 2-3 kliknutí. První pro otevření modálního okna a druhé pro přechod na danou položku, ale to jen v případě, že je zrovna zobrazené takové menu, které požaduji, jinak by to bylo o jedno kliknutí navíc. Toto přepínání mezi obrazovkami muselo jít po celý čas zobrazených stránek, aniž bych musel otevírat nějaké modální okno. Zároveň by bylo ideální přepínat na pouhý jeden klik. Proto mým druhým řešením bylo implementovat podobnou lištu, jako se nachází na všech internetových prohlížečích např. Google Chrome, Mozzila Firefox, Opera a další. Výsledné vizuální řešení je zobrazeno na obrázku číslo [11].

Obrázek 11: Lišta umožňující přepínání mezi obrazovkami

V navržené aplikaci je možné klasické přidávání nových záložek, zavírání vytvořených a přepínání mezi nimi. I přesto, že výsledné řešení vypadá skoro stejně jako u daných prohlížečů, způsob implementace a fungování je trochu jiný. Záložky na liště fungují jako normální tlačítka v menu, kdy při kliknutí na nějakou položku se načte obsah této stránky. Jinak řečeno má aplikace si nepamatuje/neukládá otevřené načtené obrazovky. Obrazovky se nanovo vykreslují po kliku na danou záložku. Sice se nejedná o nejrychlejší řešení, ale aspoň šetrné k využití paměti.

Samotná implementace je díky tomuto řešení celkem jednoduchá, stačí udržovat seznam všech otevřených lišt a zároveň informaci, která je právě zobrazena. Jediný menší problém je opět se stylováním těchto záložek. Pro jejich tvorbu jsem použil samotné divy, které jsem následně náročně styloval. Jejich stylování je rozděleno na 5 částí, kde první je hlavní střední část, další dvě jsou zkosené části na krajích. Na pozadí těchto dvou částí jsou vytvořeny další, které jsou vždy o jeden pixel vysunuté za nimi, jsou tmavší a způsobují tak tmavý rám kolem této části.

5.8 Animované kolo-loterie

Jednalo se o úlohu patřící jinému projektu a to projektu pro Den kariéry, kdy firmu Rieter navštěvují žáci základních škol. Úkolem bylo vytvořit aplikaci, která by umožňovala losování o ceny. Tato aplikace měla komunikovat se serverem a ten zase s aplikací, která běžela na firemním mobilu. Na tomto projektu jsme pracovali dva a měli jsme jasně rozdělené úlohy. Já měl na starosti právě aplikaci loterie, která měla běžet na internetovém prohlížeči. Měl jsem představu řešení, kde by se točilo klasické "kolo štěstí"a to by fungovalo tak, že by se na signál startu začalo konstantní rychlostí otáčet a po signálu zastavení zase postupně zpomalovat až do nulové rychlosti. Řešení tohoto problému nebylo snadné, a tak jsem pro usnadnění práce našel na internetu řešení, které mi ulehčilo jak práci s fyzikou, tak s vykreslováním kruhu. Nalezené řešení jsem importoval do mého React projektu a propojil.

Vykreslení kola ve staženém skriptu je řešeno canvasem a je voláno metodou `window.requestAnimationFrame(callback)`, kdy `callback` metoda vykresluje celý kruh. Tato metoda provádí update ještě předtím, než je provedeno nové vykreslení stránky. Pro uložení cen, které byly k dispozici, jsem vytvořil objekt, do kterého jsem přidal jejich název, počet dostupných a cestu k jejich obrázku. Problémem bylo, že při restartu aplikace nebo pouhém obnovení stránky se tyto hodnoty resetovaly na původní. Jako řešení jsem využil lokálního uložiště v prohlížeči, kde jsem tento objekt vytvořil a pak v aplikaci spravoval. Toto řešení právě zabraňovalo resetování hodnot objektu při pouhém obnovení stránky či celém restartu.

Implementované řešení losování bylo trochu podvodné. Vždy, když ukazatel na kole štěstí ukázal na nové políčko, zaslal informaci ze skriptu, který se o tuto animaci staral, do své React komponenty. Tato komponenta následně vždy vykreslila nový obrázek. Nicméně zobrazený výherní předmět nebyl vůbec závislý na tom, na které políčko zrovna ukazatel ukazuje, ale na pouhé pravděpodobnosti, kterou jsem vypočítal ze seznamu zbývající cen. Pro tento výpočet jsem projížděl objekt všech výherních předmětů a pokud byl jejich počet větší než 0, vždy jsem uložil cenu do nového pole a to tolikrát, kolik bylo kusů. Po takto vytvořeném poli cen jsem vzal

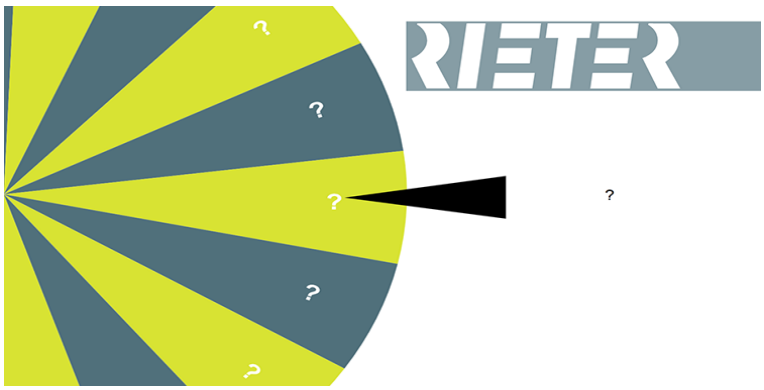
náhodnou pozici tohoto pole a vrátil zobrazený výherní předmět. Tento algoritmus je vidět na následující ukázce číslo [6].

```
let items = [];
let counter = 0;
let itemCounter = 0;
let item;
let objet = [];
for(item in retrievedObject) {
   itemCounter++;
    if (retrievedObject[item].count > 0) {
        object.push(retrievedObject[item]);
        for(let i = 0; i < retrievedObject[item].count; i++ ){
            counter++;
            items.push(itemCounter);
        }
    }
}
let random = Math.floor((Math.random() * counter));
let index = items[random];
item = object[index-1];
```

Výpis 6: Algoritmus získávání výherní ceny

Komunikace se serverem pak probíhala pomocí běžných HTTP dotazů GET/POST, kde pomocí GET metody jsem se dotazoval serveru, jestli mám kolo roztočit nebo začít losovat. POST metodou jsem potom posílal zpátky informace o aktuálním stavu loterie.

Ze začátku jsem se setkával s problémem neplynulé, až trochu trhané animace. Zkoušel jsem místo vykreslování kruhu vložit obrázek, který rotoval kolem své osy, ale rozdíl byl zanedbatelný. Problémem byla nejspíš hlavně velikost kruhu, jeho rozměry byly 1500x1500px. Tento problém jsem řešil vykreslováním pouze části kola, která byla vidět,. Toto celkem výrazně zvýšilo plynulost celé animace, kterou jsem ještě dolaďoval malými úpravami vykreslovacích funkcí. Výslednou vizualizaci této loterie lze vidět na následujícím obrázku [12].



Obrázek 12: Kolo štěstí

6 Celkové zhodnocení praxe

6.1 Uplatnění teoretických a praktických znalostí získaných ve škole

Jednoznačně nejdůležitější věcí, kterou mě vysoká škola naučila a tím připravila na tuto praxi, je zvládnutí samostudia. Ve frontendu pracuji v týmu jako jediný, tudíž jsem rady/tipy hledal sám na internetu. Od začátku praxe dělám v Reactu, ve kterém jsem nikdy předtím nedělal, používám nové architektury, které jsem nepoužíval. React je JS knihovna, takže základní znalosti JS nesměly chybět, také základy HTML a CSS byly velmi důležité. Tyto požadované základy jsem si osvojil v předmětu Vývoj internetových aplikací. Další důležitou věcí, kterou mě škola naučila, bylo programátorské myšlení, kterému jsem se učil v průběhu celého studia na vysoké škole. Velmi důležité předměty, které mě také připravily na tuto praxi, byly Algoritmy a Matematika. Bez těchto dvou předmětů bych nezvládl úlohu jako například sestrojení matice, které vyžadovalo mnoho výpočtů. Dalším užitečným předmětem byla určitě Správa systému Windows, kde nás učili pracovat a vytvářet virtuální stroje. Toto jsem potřeboval, když jsem si měl na svém notebooku spustit systém Pheonix přes VMware player , na kterém běžela předchozí desktopová aplikace.

6.2 Chybějící znalosti a schopnosti

Hlavním nedostatkem při řešení zadaných úloh v praxi, byla chybějící znalost JS knihovny React, s kterou jsme se během studia neseznámili, ačkoliv je v praxi přitom velmi rozšířená. Proto bych určitě přivítal v rámci studia mého oboru více předmětů zaměřených na webové technologie.

6.3 Získané znalosti

K získaným vědomostem bych také zařadil pokročilou znalost webových technologii, jako je třeba React, Flux.. Dále znalost CSS a HTML jazyka, zefektivnění samostudia a rozvoj programátorského myšlení.

7 Závěr

Na začátku praxe, po zadání první úlohy matice jsem měl obavy z komplexní úlohy psané ve frameworku, o kterém jsem slyšel poprvé. Navíc jsem v našem týmu jediný, kdo pracuje ve frontendu. Díky mé snaze při samostudiu a tendenci se zlepšit, jsem se s první úlohou vypořádal nad očekávání dobře. Další úlohy pak pro mě byly snadnější. Dnes už téměř nemám žádné problémy s řešením jakékoliv úlohy, kterou dostanu ve firmě zadanou.

Celkově hodnotím mou bakalářskou praxi ve firmě Rieter velmi pozitivně. Od začátku je tu skvělý kolektiv lidí, který mě mezi sebe okamžitě přijal. Velmi mě potěšilo a zároveň pozitivně motivovalo ocenění ze strany vedení firmy a projevený zájem o můj návrh menu kruhového designu. Firma mi nabídla pozici Webového vývojáře a s ohledem na mé dosavadní studium pracuji v současné době pro firmu na zkrácený úvazek zaměstnanecké smlouvy. Plánuji další spolupráci s touto firmou i po ukončení studia a těším se na ni.

Literatura

[1] Rieter History [online]. Dostupné na: http://www.rieter.com/cz/rieter/about-rieter-group/subsidiaries/rieter-cz-sro/historie/

[2] React History [online]. Dostupné na: https://en.wikipedia.org/wiki/React_(JavaScript_library)

I Obsah přiloženého CD

- soubor unz0009.pdf -> Bakalářská práce v pdf formátu
- složka photos -> Fotky strojů, na kterých poběží má aplikace
- složka screens -> Screeny obrazovek, kde bude zachycena vizualizace jak staré aplikace, tak mé nové